Apellidos y Nombre: _____

Implementación de RISCY en FPGA

Departamento de Tecnología Electrónica
Universidad de Sevilla

1 Material

- Ordenador con Xilinx ISE instalado.
- Placa de desarrollo Basys2. Incluye la FPGA Xilinx Spartan-3E.
- Ficheros con las plantillas para el diseño (lab_kit).

2 Objetivos

- Utilizar la descripción Verilog del computador RISC-Y desarrollado en clase.
- Simular el comportamiento del computador para diversos programas escritos en código máquina.
- Implementar el sistema en una FPGA.
- Probar el sistema y el código de programa a partir de la introducción de datos y la visualización del resultado.

3 Especificación del sistema

El computador RISC Y desarrollado en clase puede completarse mediante un sistema de entrada/salida muy básico que le permite interaccionar con el exterior a través de dos buses: din y dout.

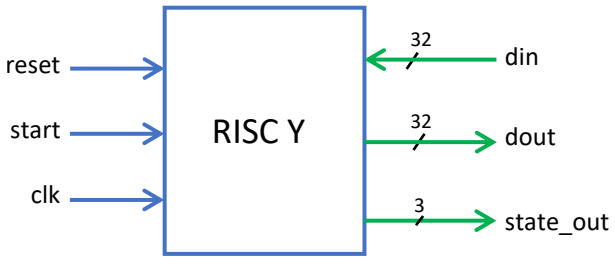

Figura 1

Para ello, en el código Verilog (riscy.v en lab_kit) ha de tenerse en cuenta lo siguiente:

1) Bus *din*: se ha conectado a la entrada del registro X15
2) Bus *dout*: se ha conectado a la salida de X14
3) Bus *state_out*: permite monitorizar el estado de la unidad de control
4) Los buses de dirección de las memorias DATMEM y CODMEM se han definido con menos de 32 bits, ya que no necesitamos tanto espacio de memoria.

En la figura 2 se muestran las conexiones que permitirán interaccionar con RISC Y a través de los elementos de la placa Basys 2. Todas ellas se explican a continuación y se han establecido en el archivo system.v (incluido en lab_kit).

1) La señal de *reset* se conecta a la AND de los botones BTN2 y BTN1, de esta forma solo si se presionan simultáneamente, se activa *reset*.
2) La señal de *start* se activa presionando el botón BTN3.
3) La entrada de datos *din* se ha conectado a los *switches* de la placa para poder dar valores de entrada. Dado que solo hay 8 *switches* solo es posible especificar los bits menos significativos, el resto de bits hasta 32 se han fijado a 0.
4) Los 16 bits menos significativos de la salida de datos *dout* se muestran en los *displays* 7-segmentos de la placa, para ello se ha introducido un controlador del *display* que realiza la conversión de la salida binaria a 7 segmentos.
5) La señal de reloj se ha conectado a un módulo *prescaler* que obtiene una señal lenta a partir del reloj de la placa para que se pueda observar ciclo a ciclo la evolución del computador.
6) El bus *state_out* se ha conectado a los leds de la placa, se ha utilizado un decodificador para que en cada estado se ilumine un único led en cada estado.

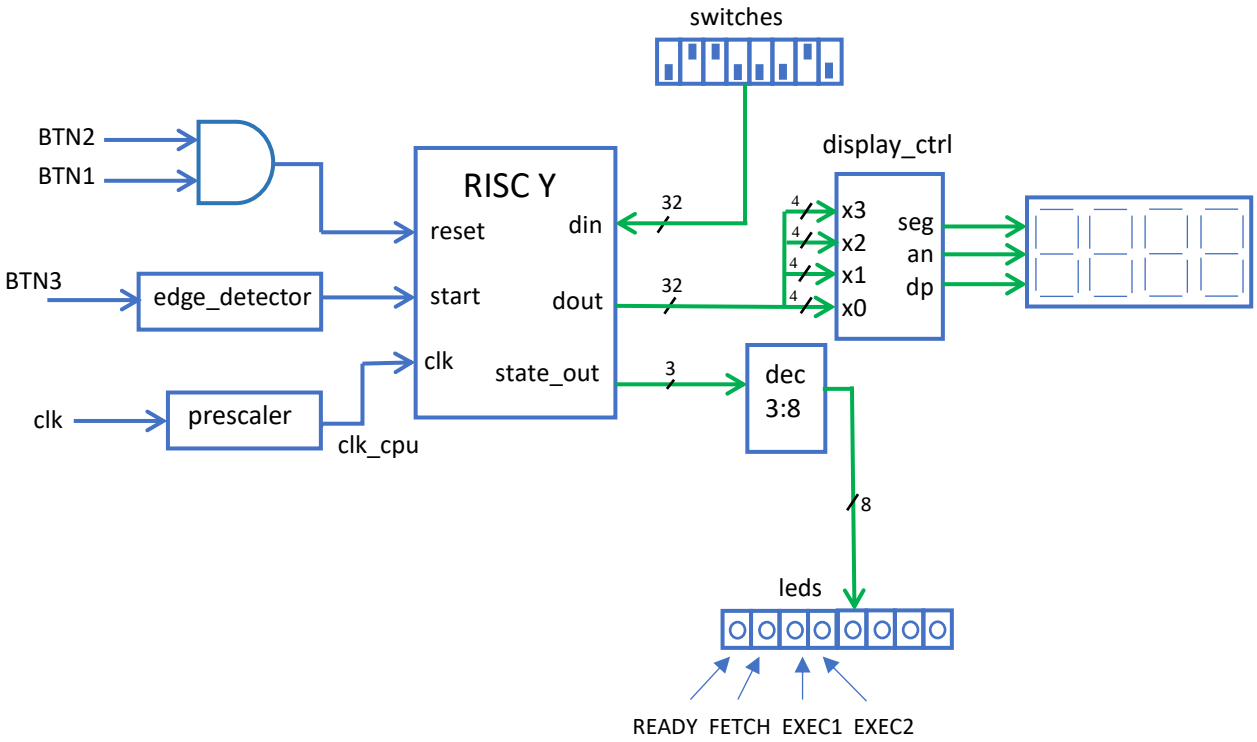

Figura 2

4 Descripción del kit de laboratorio

El kit del laboratorio se compone de los siguientes archivos:

globals.vh: definiciones globales de macros utilizadas por el resto de archivos/módulos.

data_unit.v: descripción de la unidad de datos.

control_unit.v: descripción de la unidad de control

code_mem.v: memoria de programa.

alu.v: descripción de la unidad aritmético-lógica.

riscy.v: conexión entre las instancias de la unidad de control y la unidad de datos.

riscy_tb.v: banco de pruebas (test bench) para el procesador RISC-Y.

system.v: descripción del sistema de la figura 2 (conexión con los elementos de la placa).

display_ctrl.v: archivo que controla el display 7 segmentos de la placa.

Basys2_100_250General.ucf: archivo de restricciones de síntesis para la placa Basys 2

5 Trabajo de laboratorio

Antes de realizar este apartado debe haber leído y comprendido el apartado 3 donde se dan las especificaciones del sistema que se va a implementar.

Debe crear un proyecto como habitualmente utilizando el entorno Xilinx ISE y añadir todos los archivos correspondientes al computador RISC Y que se le proporcionan en el lab_kit.

Las propiedades del proyecto adecuadas para la placa Basys 2, son:

- General Purpose
- Family: Spartan 3E
- Device: XC3S100E
- Package: CP132
- Speed grade: -5

5.1 Simulación del programa de prueba

Debe comprobar que los archivos cargados funcionan adecuadamente. Para ello debe localizar el programa que aparece escrito en la memoria de código (archivo code_mem.v).

El programa está escrito utilizando las macros del archivo globals.vh que puede consultar en el anexo y que son traducidas su equivalente código máquina.

Este programa realiza la siguiente tarea: dado un valor de entrada (N) obtiene la suma N+(N-1)+(N-2)+…+1. Sus instrucciones son:

```
addi x14,x0,0
add x13,x15,x0
add x14,x13,x14
addi x13,x13,-1
bne x13,x0,8    → el salto es a la dirección 8 (3ª instrucción: add x14,x13,-1)
stop
```

Como sabe, cada instrucción ocupa 32 bits en código máquina y las direcciones son direcciones de bytes, por tanto las instrucciones del programa anterior ocupan las direcciones 0,4,8,12,16,20 (en decimal) o 0,4,8,c,10,14 (en

hexadecimal). Todas estas direcciones son múltiplos de 4 (en binario acaban en 00).

Por este motivo en el código Verilog de la memoria de código no se han usado esos bits y las posiciones usadas aparentemente son 0,1,2,3,4,5. Es solo apariencia, internamente las direcciones son esas multiplicadas por 4. Por eso, el salto *bne* es a la dirección 8 (y no a la 2).

Simule y compruebe que efectivamente en los buses *din* y *dout* de la simulación aparecen los datos correctos.

5.2 Implementación en BASYS 2

Implemente el diseño de prueba en la placa de desarrollo Basys2.

Para ello, edite el archivo UCF y realice las conexiones adecuadas de las señales del diseño con la placa. Tendrá que descomentar las líneas correspondientes al reloj MCLK, las del display (seg<0:6>, dp y an<3:0>), todos los leds, todos los switches y las correspondientes los botones BTN3, BTN2 y BTN1.

Ejecute el proceso de síntesis e implementación del diseño para obtener el archivo de configuración de la FPGA (*bitstream*). Para ello seleccione el modo *Implementation* y ejecute los pasos *Synthesize – XST*, *Implement Design* y *Generate Programming File*. Haga las correcciones necesarias en el caso de que haya errores.

Programe el diseño en la placa. Para ello, conéctela mediante el puerto USB, asegúrese de tener accionado el conmutador de encendido y ejecute la herramienta Adept. Cargue el archivo *system.bit,* que se ha generado en el paso anterior.

Compruebe que el circuito funciona. Para ello, en primer lugar debe accionar los botones 1 y 2 (reseteo inicial) y posteriormente el botón 3 (start). Para distintos valores de entrada observe el display de salida. Cada vez que cambie el valor de entrada debe accionar el botón 3 (start).

6 Simulación e implementación en BASYS 2 de otros programas

Realice, simule e implemente otros programas. Recuerde al escribir los programas que *din* (switches) se conecta a X15 y *dout* (display) se conecta a X14.

1) Programa que lee un dato lo compara con 64 y muestra en salida el menor de los dos.
2) Programa que lee un dato y proporciona consecutivamente los 10 primeros múltiplos del número en la salida.
3) Programa que lee un dato y proporciona su producto por un número fijo.
4) Programa que calcula N términos de la sucesión de Fibonacci. N es el dato introducido en din.

7 Anexo: macros definidas en globals.vh

```
`define __RYGLOBALS
//// General definitions
`define DATA_START  12'h0
`define DATA_END    12'h0ff
//// Assembly operation codes
`define ADD     8'h01
`define SUB     8'h02
`define ADDI    8'h03
`define LUI     8'h04
`define STOP    8'h05
`define LW      8'h06
`define SW      8'h07
`define J       8'h08
`define B       8'h09
`define XOR     8'h0a
`define OR      8'h0b
`define AND     8'h0c
`define XORI    8'h0d
`define ORI     8'h0e
`define ANDI    8'h0f
`define SLL     8'h10
`define SRL     8'h11
`define SRA     8'h12
`define SLLI    8'h13
`define SRLI    8'h14
`define SRAI    8'h15

//// Registers
`define X0      4'h0
`define X1      4'h1
`define X2      4'h2
`define X3      4'h3
`define X4      4'h4
`define X5      4'h5
`define X6      4'h6
```

```
`define X7      4'h7
`define X8      4'h8
`define X9      4'h9
`define X10     4'ha
`define X11     4'hb
`define X12     4'hc
`define X13     4'hd
`define X14     4'he
`define X15     4'hf

//// ALU operation codes
`define ALU_ADD  4'd0
`define ALU_SUB  4'd1
`define ALU_TR2  4'd2
`define ALU_XOR  4'd3
`define ALU_OR   4'd4
`define ALU_AND  4'd5
`define ALU_SLL  4'd6
`define ALU_SRL  4'd7
`define ALU_SRA  4'd8

//// Status register flag positions in funcs
`define EQ      4'd0
`define NE      4'd1
`define LT      4'd2
`define GE      4'd3
`define LTU     4'd4
`define GEU     4'd5
```